Thesis no: MECS-2014-09

Quantifying the noise tolerance of the OCR engine Tesseract using a simulated environment

Henrik Nell

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Game and Software Engineering. The thesis is equivalent to 20 weeks of full-time studies.

Contact Information:
Author(s):
Henrik Nell
E-mail: spidermine1@gmail.com

University advisor:
Dr. Siamak Khatibi
Department of Communication Systems

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Context. Optical Character Recognition (OCR), having a computer recognize text from an image, is not as intuitive as human recognition. Even small (to human eyes) degradations can thwart the OCR result. The problem is that random unknown degradations are unavoidable in a real-world setting.
Objectives. The noise tolerance of Tesseract, a state-of-the-art[1] OCR engine, is evaluated in relation to how well it handles salt and pepper noise, a type of image degradation. Noise tolerance is measured as the percentage of aberrant pixels when comparing two images (one with noise and the other without noise).
Methods. A novel systematic approach for finding the noise tolerance of an OCR engine is presented. A simulated environment is developed, where the test parameters, called test cases (font, font size, text string), can be modified. The simulation program creates a text string image (white background, black text), degrades it iteratively using salt and pepper noise, and lets Tesseract perform OCR on it, in each iteration. The iteration process is stopped when the comparison between the image text string and the OCR result of Tesseract mismatches.
Results. Simulation results are given as changed pixels percentage (noise tolerance) between the clean text string image and the text string image the degradation iteration before Tesseract OCR failed to recognize all characters in the text string image. The results include 14400 test cases: 4 fonts (Arial, Calibri, Courier and Georgia), 100 font sizes (1-100) and 36 different strings ($4 \cdot 100 \cdot 36 = 14400$), resulting in about 1.8 million OCR attempts performed by Tesseract.
Conclusions. The noise tolerance depended on the test parameters. Font sizes smaller than 7 were not recognized at all, even without noise applied. The font size interval 13-22 was the peak performance interval, i.e. the font size interval that had the highest noise tolerance, except for the only monospaced font tested, Courier, which had lower noise tolerance in the peak performance interval. The noise tolerance trend for the font size interval 22-100 was that the noise tolerance decreased for larger font sizes. The noise tolerance of Tesseract as a whole, given the experiment results, was circa 6.21 %, i.e. if 6.21 % of the pixel in the image has changed Tesseract can still recognize all text in the image.

Keywords: Optical Character Recognition, salt and pepper noise, Tesseract

Acknowledgments:

- Dr. Siamak Khatibi, for sharing invaluable insights.
- Viktor Sidén, for not plotting against me.
- Mattias Liljeson, for lending a laptop at the presentation on 2014-05-28.

Contents

Chapter 1

Introduction

Optical Character Recognition (OCR) is the recognition and conversion of characters in a digital image to digital text. This thesis focuses on machine-printed Latin alphabet characters or equivalent, as opposed to hand-written characters.

Commercial OCR dates back to the 1950s[2]. Notwithstanding decades of development, OCR is not trivial, as put by Kae et al in 2010: "getting optical character recognition (OCR) systems to consistently obtain very high accuracy rates continues to be a challenging problem"[1].

Hard copy documents, such as paper, hold much of the world's information.[3] An OCR engine can make the text of a scanned document searchable, as exemplified by the Google Books Library Project, a large-scale (more than 20 million books scanned as of 2013[1]) book scanning endeavor[4].

1.1 Document image degradation

An OCR engine may fail to recognize characters due to document image degradation.[5] Baird[6] defines "degradations" as "every sort of less-than-ideal properties of real document images". Ye and Doermann[7] expand on Baird's definition, adding: "that reduce the information or visual quality with respect to the original source", and identify 4 document image degradation sources in the document image generation process, namely: creation, external degradation, digitization and processing. They list specific degradation types as occurring in each stage:

Creation: paper texture and translucency, inadequate or heavy printing, noise in electric components, typesetting imperfections.

External degradations (also called "physical noise"[8]): paper aging, stains, thorn-off regions, scribble, carbon copy effect, scratches and cracks.

Digitization: digitizing operations, hardware defects, paper positioning variations, pixel sensor sensitivity variations, vibrations or other motions.

Processing: binarization noise, compression noise, network data loss.

[1] `http://www.nytimes.com/2013/11/15/business/media/judge-sides-with-google-on-book-scanning-suit.html`

1.2 Aim and objectives

The goal of this thesis is to quantify the recognition failure conditions of the OCR engine Tesseract when trying to recognize degraded images; quantify how well Tesseract handles image degradation.

- Objective 1: Implement a program capable of executing test cases according to selected parameters and an image degradation model.
- Objective 2: Explore the parameter range through a controlled experiment of the test cases.
- Objective 3: Analyze the result of the controlled experiment to quantify the recognition failure conditions of Tesseract.

1.2.1 Contribution

A novel statistical simulation method for evaluating the performance of an OCR engine when subjected to degraded images is presented (refer to section 3.2).

1.2.2 Research question

How much (quantification) image degradation can the OCR engine Tesseract handle on a Latin alphabet string before failing to recognize at least one character in the string, i.e. what are the recognition failure conditions of Tesseract when OCR input images are subjected to an implementation of image degradation modeling?

1.3 Motivation

In a real-world application, such as scanning a hard copy document, it is not known beforehand how the image is and will be degraded. Furthermore: "Most denoising algorithms depend on knowing the noise level"[9]. To better prepare an OCR engine for the real world, the OCR engine should be expected to recognize characters even when they are subjected to degradation. Testing and quantifying how well an OCR engine handles the image degradation problem would allow OCR algorithm implementors to show a quantifiable improvement of OCR, by doing a test before and after an algorithm change, and then comparing the results, in a effort to show a quantifiable OCR algorithm improvement when tackling the image degradation problem.

1.3.1 OCR engines

There exist a number of available OCR engines, e.g. ABBYY FineReader (commercial), GOCS (open source), Transym (commercial) and Tesseract (open source). Chattopadhyay et al[10] state that "Tesseract is considered one of the most accurate free software OCR engines currently available". They compare Tesseract to Abby FineReader and GOCS and conclude that Tesseract has better accuracy, when doing OCR on video frames. Tesseract was developed at HP Labs 1984-1994, competing in an OCR accuracy competition in 1995, where it "shone brightly with its results"[11]. Tesseract was released open source in 2005, and is since 2006[2] sponsored by Google[3]. Considering that Google is conducting the large-scale Google Books Library Project (where OCR technology is needed), and that Tesseract is open source, state-of-the-art[1] and comparable to a commercial OCR engine[12], elucidate Tesseract as a relevant object of study.

Tesseract architecture[11]

When doing OCR, Tesseract first performs a connected component analysis to find outlines, which nest together into blobs, which contain text lines and regions. How characters are broken into words depend on if the font is monospace (fixed pitch) or proportional. This font distinction is found through analysis of the lines and regions stored in the blobs. Monospaced (fixed pitch) characters are broken into words immediately, whereas proportional words are found using definite spaces and fuzzy spaces. Recognizing the words are done using a two-pass approach. In the first pass, one word at a time is tested for recognition. A word that is deemed satisfactory is used as training data for an adaptive classifier, the idea being that more recognized words (training) improve subsequent recognition. In the second pass, all words are tested for recognition again, leveraged by the learning ability of the adaptive classifier. As a last step, lowercase characters are tested for recognition using fuzzy spaces and x-height.

1.4 Scope

The degradation modeled in this thesis is salt and pepper noise, occurring in the "Digitization" stage (refer to section 1.1). One reason for using salt and pepper noise in this thesis is its simplicity (intuitive to understand), allowing it to act as an image degradation reference test for future work, which could involve testing more OCR engines than Tesseract.

[2] `http://googlecode.blogspot.se/2006/08/announcing-tesseract-ocr.html`
[3] `https://code.google.com/p/tesseract-ocr/`

1.5 Salt and pepper noise

Salt and pepper noise on an image manifest itself as randomly occurring black and white pixels[13, p. 272]; some causes include: defective image sensor elements or dust particles inside an image sensor[14], i.e. missing pixel color data. Putting it in a wider perspective, not only image sensors may cause missing pixel color data. Consider unreliable data transmission of an image where some data may be lost; the lost image data most be substituted for something when rendered, which may cause "randomly occurring black and white pixels" (depending on the implementation of the image rendering program), which is the definition of salt and pepper noise.

Chapter 2

Related Work

Polastro et al[15] propose a method for OCR error evaluation on information retrieval in a computer forensics context, using the software "AccessData FTK" (Forensic Toolkit), which has Tesseract as OCR component. They study three image degradation types: downsampling, skews and salt and pepper noise, when trying to recognize English and Portuguese language text. Their method is based on comparing the returned number of searched words with the 100 % ideal (where all searched words are found). They print documents and subsequently scan them. Their test data, document texts, originate from a corpus consisting of 50 Portuguese newspaper articles; they used automatic translation (Google Translate) of the text from Portuguese to English, in a way skewing the "real" quality of the data used. Further, they assure that the "printed texts were carefully scanned in order to avoid the inclusion of noise and skew", without specifying what actual measures were taken to prevent noise inclusion, if at all possible. They use only one font: Times New Roman, and one font size: 12, and two levels of salt and pepper noise: 1 % and 3 %, without specifying what the measures mean. They conclude that 3 % salt and pepper noise have a "great influence" on the OCR result (refer to section 5.1).

In comparison, this thesis propose a novel method of simulating salt and pepper noise using a systematic set of parameters (not only one font and font size), using a simulated environment acting as a controlled experiment where noise is not introduced inadvertently (and with unknown quantified effect) by scanning. Further, the salt and pepper noise measurement is defined and its application is fine-grainly controlled.

Bieniecki et al[16] mention an earlier work of theirs, written in Polish, where they "ran a simulation of OCR detection under controlled influence of noise, geometric distortions, varying image resolution and lighting conditions.", measuring OCR accuracy of FineReader 7.0 when subjected to different lighting conditions, geometric distortions and varying image resolutions. Bieniecki et al used, in contrast to Polastro et al[15], a digital image simulating the ideal condition, instead of a scanned document. This ideal condition was simulated as a Microsoft Word document image saved as a 600 dpi bitmap, which was then degraded using the aforementioned approaches. They simulated low-lighting conditions as Gaussian

noise. Bieniecki et al draw a number of conclusions, including that the OCR result "was not much hampered by uneven lighting" and that OCR accuracy drops when lowering the resolution below 300 dpi. Further, they conclude that OCR accuracy was most affected by geometric deformations.

The second presented related work relate to this thesis in that it uses digital images, instead of scanned documents, when measuring the impact of OCR in relation to noise. OCR accuracy may be measured without explicitly taking noise into account. Such related work are presented below.

Patel et al[12] compare Tesseract with another OCR engine: Transym. They photograph twenty vehicle number plates and use Tesseract and Transym to extract the number plate data (alphanumeric characters). Their experiment explores the OCR accuracy when using gray scale or color images of the number plates. The accuracy is measured by comparing the number of recognized characters to the total number of characters on the number plates. They conclude that Tesseract is both faster and more accurate than Transym, for both color images and grayscale images. Their paper lacks, however, motivation of the datasets.

Dhiman and Sing[17] compare Tesseract with GOCR under a number of conditions, including image type (color, grey scale, black-and-white), resolution (75, 300, 600, 1200 dpi), font (Arial, Roman, Tahoma) and brightness value setting when scanning; they print document and scan them, however they forget to mention what the "documents" are, i.e. the dataset is not defined. They mention that 39 characters are used in total, but not how they are divided across the test conditions. Further, the experiment variables, e.g. the fonts used, are not motivated. They conclude that Tesseract has better accuracy than GOCR in most of their tested cases.

2.1 Image noise

The literature provide papers focusing on noise removal/reduction/filtering, e.g. reducing salt and pepper noise[18][19]. Another related area of study is measuring image noise, e.g. [20] [21].

2.2 Image degradation databases

Baird describes a parameterized "defect model"[22]. In a follow-up paper[23], he gives a preview of a large public-domain character images database with ground truth and defect model parameters, "the first of its kind", called *Bell Labs image defect model database, version 0*, available on CD-ROM. Another image database, including images degraded using Baird's work, is presented by Guyon et al[24].

Chapter 3

Method

3.1 Method selection

This thesis uses a quantitative research method through an implementation of an experiment, to allow for a controlled and systematic study of the studied phenomenon: OCR noise tolerance. The reason for not using a qualitative study involving humans is that OCR does not consider if humans consider a string to be correct, instead, ground truth is used to determine if an OCR result is correct or not.

3.1.1 Real versus synthetic data

To find out the answer of the research question (refer to section 1.2.2) one could use real data, such as scanned documents, or synthetic data, such as generated documents. The problem with real data is the lack of ground truth. This problem could trivially be overcome by printing a text document and scanning it (as was done in some of the related works, refer to section 2). In this case the digital text document is the ground truth, which can be compared against the OCR result in order to find out how accurate the OCR was. The problem with this approach, however, is the lack of control of all variables in the experiment. For example, how does one adjust the amount of image degradation present on the hard copy document and resulting scanned image? One could find a way to measure the amount of image degradation (as is done in [20][21]), but not adjusting it in a quantifiable manner, without adding a synthetic touch to the real data. And how would one get a completely clean image with real data? Every scanning operation introduces artifacts to the image. Because of these matters, synthetic data is more preferred than real data. With synthetic data, a quantifiable image degradation can be applied to a clean image, making the research question answerable.

3.1.2 Existing synthetic data versus data creation

Section 2.2 mentioned OCR document image databases containing synthetically degraded characters. The idea for such a database is to cover a broad range of

languages, font types, document types etc.. The focus of this thesis, however, is to allow fine-grained control of the salt and pepper noise application, as will be described in the next section.

3.2 Experiment

The experiment idea is to, for a number of test cases, iteratively degrade a synthetically created text string image (an image containing a black string on a white background). In each iteration, the text string image is input to Tesseract for OCR, until Tesseract fails, marking the end of the current test case. Tesseract OCR failure is defined as a mismatch between the ground truth (the image text string) and the OCR result of Tesseract.

Each performed test case result in a quantification of the applied degradation present in the image, the iteration before Tesseract OCR failed, representing the maximum amount of degradation possible without affecting the Tesseract OCR result. Defining appropriate test cases, i.e. parameters used in the experiment, allow an experimenter to provide an answer to the research question posed in section 1.2.2.

3.2.1 Experiment parameters

Objectives, steps pursued in order to answer the research question, are mentioned in section 1.2. Objective 1 mention the implementation of a program capable of executing "test cases"; the test cases are meant to capture OCR-affecting aspects. In this experiment, a test case is defined as a combination of font size, font and text string. For the sake of discussion, each constituent of a test case, i.e. font size, font or text string, is referred to as a "test case element". Refer to table 3.1 for test case examples.

Id	**Font size**	**Font**	**Text string**
#1	15	Calibri	TESSERACT
#2	15	Arial	TESSERACT
#3	16	Arial	TESSERACT
#4	16	Arial	tesseract
#5	32	Courier	DEGREEPROJECT
#6	32	Georgia	DEGREEPROJECT

Table 3.1: Test case examples. Note that changing one of the test case elements creates a new test case, compare e.g. test case #1 with #2, where the font differs, thus making them separate test cases. Also note that lowercase and uppercase character are not considered equal, i.e. test case #3 and #4 are different test cases.

Objective 2 mention an exploration of the "parameter range" through "controlled experiment of the test cases". To allow the latter to be realized, the parameter range, i.e. the studied test case element values must be determined.

Font selection

The fonts were chosen to include at least one monospaced font, proportional font, serif font and sans-serif font. Refer to table 3.2.

Font	**Proportional**	**Monospace**	**Serif**	**Sans-serif**
Arial	X			X
Calibri	X			X
Courier		X	X	
Georgia	X		X	

Table 3.2: The properties of the fonts used in the experiment.

Font size selection

A font size interval of 1-100 was selected to capture the print-sized characters (around 10-30; as a reference: this text is font size 12) and also expand beyond them to be able to explore boundaries of Tesseract outside print-sized characters. Figure 7.1, 7.2, 7.3 and 7.4 (refer to appendix) illustrate both the fonts used in the experiment and the largest font size tested: 100.

Text string selection

The philosophy was to include strings with:

- different number of characters.
- all letters in the Latin alphabet (A-Z, a-z).
- both dictionary words and made-up words (Tesseract has some form of linguistic analysis[11], meaning this distinction could have an effect on the OCR result).

The strings used in the experimented are presented and indexed below:
(1) "TESSERACT", (2) "OPENCV", (3) "DEGREEPROJECT", (4) "ABC", (5) "COMPUTER", (6) "INSTITUTE", (7) "TECHNOLOGY", (8) "DEFY", (9) "INTERNATIONAL", "(10) BOAT", "EXPANSION" (11), "HY" (12), "JUMBO" (13), "Z" (14), (15) "VIOLENT", (16) "WORM", (17) "QUICK", (18) "XAM", (19) "tesseract", (20) "opencv", (21) "degreeproject", (22) "abc", (23) "computer", (24) "institute", (25) "technology", (26) "defy", (27) "international", (28) "boat", (29) "expansion", (30) "hy", (31) "jumbo", (32) "z", (33) "violent",

(34) "worm", (35) "quick", (36) "xam".

The listed strings are the dataset of the experiment. The reason for testing more than one string is to simulate a real-world scenario. It is important to have unrelated text strings, otherwise, if two strings are highly related, the additional string provide less new information to the experiment, compared to if the strings had been completely unrelated. The relatedness (correlation) between all strings to each other is calculated (refer to figure 7.6 in appendix) and presented in figure 7.5 (refer to appendix). The correlation of a string to itself is 1, whereas a correlation of 0 means the two strings are not correlated at all, which is the ideal case in this experiment. The highest correlation between any two strings (except for each string with itself) from the dataset is circa 0.67, e.g. between string (1) and (6), i.e. the correlation between each string and each other string is never larger than 0.67.

At least 50 % uncorrelatedness is granted when using both uppercase and lowercase characters for the each "word", compare e.g. (1) and (19). About 60 % (388 out of 648) of the correlation calculations show a correlation of 0, i.e. circa 60 % of the strings are completely uncorrelated.

3.2.2 Experiment setup

Each test case is performed 18 times, i.e. each test case produces 18 noise tolerance measurements. These measurements are assumed to be normally distributed. Doing each test case more than one time and calculating the mean gives a more accurate (lessened impact of randomness) result than doing each test case one time. The number 18 was determined using a sample size determination formula from a book named "Business Research Methods"[25, p. 436]:

$$n = \frac{Z_c^2 pq}{E^2} \tag{3.1}$$

where
n = number of items in sample
$Z_c^2 pq$ = square of the confidence level in standard error units
p = estimation of proportion of successes
q = 1 - p, estimated proportion of failures
E^2 = square of the maximum error allowance

Using a confidence interval of 95 % gives a $Z_c^2 pq$ of 1.96, according to the book source. Using a p value of 0.5, the stated confidence interval, and an E value of 0.23:

$$n = \frac{1.96^2 \cdot 0.5 \cdot 1 - 0.5}{0.23} \simeq 18 \tag{3.2}$$

Salt and pepper noise application (pseudocode)

```
Loop for each image pixel:
    Generate uniformly distributed random number in the interval -1.0 to 1.0
    If random number is greater than 0.99:
        Set pixel color to white
    Else if random number is smaller than -0.99:
        Set pixel color to black
Loop end
```

Salt and pepper noise measurement

The amount of salt and pepper noise present in an image is measured as the changed pixels percentage (henceforth referred to as "noise tolerance"), when comparing the clean input image with the degraded image the iteration before Tesseract OCR failed.

$$\frac{c}{m \cdot n} \cdot 100 \tag{3.3}$$

Equation 3.3: Noise tolerance when comparing two equally-dimensioned images. Where
m = image width
n = image height
c = number of changed pixels.

For example, consider a 200x300 pixels white image ("A") containing a text string in black. A copy of the image ("B") is synthetically degraded with salt and pepper noise. After Tesseract OCR failure, the pixels from "A" are compared to corresponding pixels in "B", i.e. the pixel at e.g. coordinate (20, 25) in "A" is compared to the pixel at coordinate (20, 25) in "B"; this comparison is done for every pixel coordinate. Assuming, for the given example, that 2400 pixels were found to be different, gives a noise tolerance of 4 % (2400/(200·300)·100).

The image dimensions depend on the font size of the image string, due to the fact that images are cropped to include as little whitespace as possible, in order to focus the salt and pepper noise to the text string.

Tesseract's adaptive classifier

The idea for the experiment is that test cases should be independent of each other, i.e. the order of test case execution should not affect the experiment result. Utilizing Tesseract's adaptive classifier would defeat this goal. Smith (the original creator of Tesseract) states, about the adaptive classifier of Tesseract: "Each word that is satisfactory is passed to an adaptive classifier as training data. The adaptive classifier then gets a chance to more accurately recognize text lower down

the page."[11], implying different results if the page had been read bottom to top instead of top to bottom. In the experiment program, presented as pseudocode below, Tesseract's adaptive classifier is reset after each OCR operation, to ensure the independency of each test case.

Experiment program[1] (pseudocode)

```
1  Loop through test cases:
2      Loop each test case 18 times:
3          Create large image in memory and render test case string to it
4          Crop the image, leaving no whitespace outside the utmost character edge
5          Loop until Tesseract OCR failure:
6              If not first loop iteration
7                  Apply small amount of random salt and pepper noise to the image
8              Input image to Tesseract
9              If Tesseract OCR failed
10                 Save noise tolerance measurement to result file
11             Reset Tesseract's adaptive classifier[2]
12         Loop end
13     Loop end
14 Loop end
```

[1]The program is written in C++. The open source computer vision library OpenCV (http://opencv.org/) is used for image manipulation.

[2]The baseapi.h source code file of Tesseract, has the following comment about the "ClearAdaptiveClassifier" function: "Call between pages or documents etc to free up memory and forget adaptive data"

3.2.3 Experiment illustration

This section illustrates one of the test cases performed by the experiment program. Refer to figure 3.1

Figure 3.1: Illustration of a test case executed by the experiment program. Font size: 40, font: Arial, text string: DEGREEPROJECT. This particular test case required 7 degradation iterations (shown top-down) before Tesseract OCR failed. The topmost text string image (first degradation iteration) is clean, i.e. without degradation. Subsequent text string images are degraded, using the previous text string image as source, i.e. the third text string image, for example, is the second text string image with more degradation applied. The bottom text string image is the degradation iteration where Tesseract OCR failed. The failed OCR result was: "DEGREEPROJECLT", and the noise tolerance measurement was circa 2.73 %.

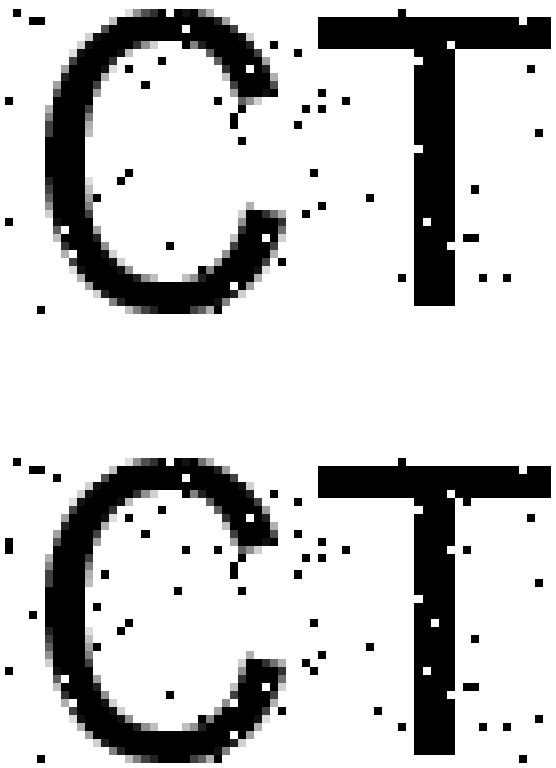

Figure 3.2: Enlarged excerpt from figure 3.1, showing the last two characters ("CT") from degradation iteration 6 (top) and 7 (bottom). Tesseract recognized the top text string image as "CT", whereas the bottom text string image was recognized as "CLT".

For the test case illustrated in this section, it appears a few pixels in the area between "C" and "T" (refer to figure 3.2) have changed (due to salt and pepper noise) between degradation iteration 6 and 7 (refer to figure 3.1), causing a Tesseract OCR failure.

3.2.4 Experiment execution

As stated in section 3.2.1: 4 fonts, 100 font sizes (interval: 1-100) and 36 strings were selected as experiment parameters. 14400 (4·100·36) test cases were executed by the experiment program, taking about 19 hours to complete. Each test case was tested 18 times, with an unknown number of subiterations taking place before Tesseract OCR failed; about 1.8 million OCR attempts were performed by Tesseract during the execution of the 14400 test cases.

Chapter 4

Results

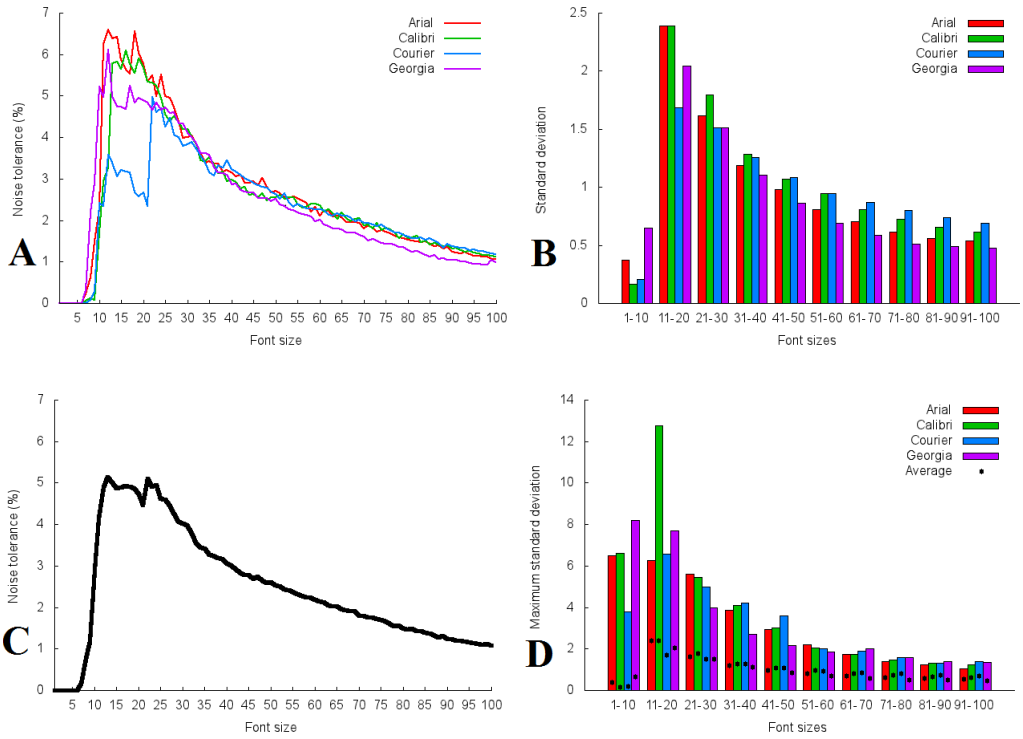

Figure 4.1: **(A)** Noise tolerance for 36 strings and 4 fonts for the font size interval 1-100. The noise tolerance measurements from the 18 test cases for each font size are averaged, and then this averaged value is averaged again for the 36 strings. **(B)** Standard deviation of the noise tolerance measurements presented in (A). Standard deviation values have been averaged for the 18 test cases, and then averaged again into intervals with 10 font sizes in each. **(C)** The noise tolerance of the 4 fonts from (A) is averaged into one line. **(D)** Maximum value of the standard deviation in the given font size interval is presented as bars. Average standard deviation from (B) is shown as black dots on the bars.

Chapter 5

Analysis

For font sizes smaller than 7, Tesseract does not recognize any of the test strings (refer to figure 4.1C), even when they are not subjected to noise. This could be due the smallness of the characters, i.e. the unique characteristics used by OCR to identify each character is not yet recognizable, causing an OCR failure. In the same figure it is noted that the noise tolerance trend is increasing from font size 7, peaking at font size 13 (circa 5.14 %), where it starts a small decline until font size 21 (circa 4.46 %). This small decline trend is not relevant, considering that the standard deviation in the font size interval 11-20 is 2.13 (refer to figure 5.1), i.e. the small trend could have happened by chance. A relevant trend is visible from font size 22 to font size 100. The trend is: decreasing noise tolerance for larger font sizes. A reason for this could be polygonal approximation used by Tesseract, mentioned as its "key weakness" by its original author Ray Smith [11].

Four fonts were tested: Arial, Calibri, Courier and Georgia. They behave mostly the same (refer to figure 4.1A), i.e. noise tolerance is mostly the same for each of the fonts. The exception is Courier, which has lower noise tolerance than the other three fonts, in the font size interval 13 (where the peak is for the average graph, figure 4.1C) to 22. An explanation to why Courier behaves differently than the rest of the fonts is that Courier is monospaced and the other three fonts are proportional. It was mentioned in section 1.3.1 that Tesseract handles monospaced fonts differently than proportional fonts. The peak performance interval overlaps with what in this thesis has been called "print-sized characters" (refer to section 3.2.1).

The peak performance interval is identified as the interval having the highest average noise tolerance. When comparing figure 4.1A and figure 4.1C in the font interval 13 to 22 it can be observed that Courier decreases the average, which have two peaks, font size 13 (circa 5.14 %), and font size 22 (circa 5.11 %). The test data suggests that Tesseract is better suited to handle proportional fonts in the delineated peak performance interval, as seen in figure 4.1A.

The standard deviation of the noise tolerance measurements follow the same trend as the noise tolerance measurements themselves, as seen when comparing figure 4.1A and figure 4.1B, i.e. the noise tolerance trend following font size 21 gets more reliable the for larger font sizes.

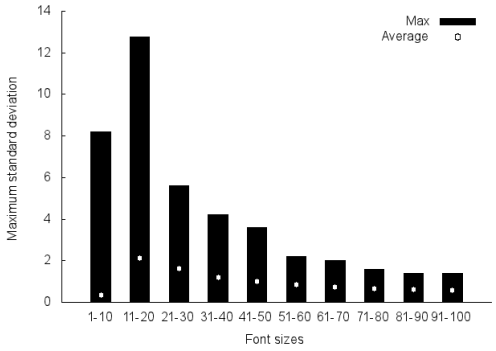

Figure 5.1: Maximum of the standard deviation measurements in each interval, averaged for the 4 tested fonts. Standard deviation (as was presented in figure 4.1B) is shown as a white dot on each bar, averaged for the 4 tested fonts.

The standard deviation for the font size interval 11-20, is circa 2.14 (refer to figure 5.1), higher than the other font size intervals in the experiment, implicating higher measurement uncertainty for the peak performance interval. The highest maximum standard deviation from the experiment, circa 12.75, is also found in the peak performance interval, refer to font size interval 11-20 (figure 5.1).The highest noise tolerance is found in the peak performance interval, but the measurements also deviate from the mean more than any other intervals tested, as shown in figure 4.1B. Courier, however, had both the lowest standard deviation average (refer to figure 4.1B) and maximum standard deviation (refer to figure 4.1D) of the fonts in the peak performance interval, implying that it is likely to be worst performing in the peak performance interval.

5.1 Synthesis

Polastro et al[15] measure OCR error using edit distance: the number of operations (deletions, insertions or replacements) needed to transform the OCR result string to the correct string. This thesis uses, given this information, an edit distance of 1 or higher, as stopping criterion, instead focusing on measuring noise, when the stopping criterion is met. Further, they test only one font, Times New Roman, and one font size: 12. They scan printed papers, inadvertently introducing image degradation before applying synthetic image degradation. Only two levels of salt and pepper noise are used: 1% and 3%; it is not explicitly stated exactly what these two levels mean. They conclude the salt and pepper experiment with: "Image degradation, in particular the application of salt-and-pepper 3% noise, had a great influence on the results obtained." As for this thesis and font size 12, 3% salt and pepper noise does not have a "great influence" on the OCR result for any of the fonts tested, even though Courier is almost affected

(refer to figure 4.1A). They measure on whole documents, however, and not on single words, as is done in this thesis.

This thesis notes that larger font sizes have lower noise tolerance than smaller font sizes. However, as previously stated (refer to section 3.2.2), image dimensions grow with font sizes, due to the tight image cropping employed in the experiment described in this thesis. Given the results of Polastro et al, it could be that the lower noise tolerance for larger fonts is caused by the larger image dimensions, not by the font size itself. It is assumed that the documents used by Polastro et al result in larger images than the ones used in this thesis, thereby bridging the gap between their work and this thesis. To clarify: it is not the image dimensions themselves that affect the recognition ability of Tesseract when handling salt and pepper noise; it is the measure of salt and pepper noise (the noise tolerance measurement) that is affected, since it depends on the image dimensions.

The differing results could also be explained by the different datasets used. Polastro et al use real scanned documents, where degradation already exists in some form, whereas this thesis uses a pristine white background. Also, they use whole documents, introducing more misrecognition possibilities, especially spaces between words, which can easily be "overwritten" by salt and pepper noise.

Chapter 6

Conclusions and Future Work

A program was implemented in order to explore the recognition failure conditions of Tesseract, resulting in noise tolerance measurements for appropriate experiment parameters defined as test cases: font, font size and text string.

From the analysis it has been observed and discussed, and is now concluded, that the recognition failure conditions of Tesseract are affected by the font sizes used for the dataset strings, and to some degree, the font used. Font sizes smaller than 7 could not be detected at all, even when not subjected to noise. The peak performance interval, the font size interval that had the highest noise tolerance, was found in the analysis to be font sizes from 13 to 22. The noise tolerance trend for larger font sizes, i.e. the font size interval 22-100, was that the noise tolerance decreased. It was found in the analysis that the one monospaced font tested, Courier, had lower noise tolerance then the other three fonts in the peak performance interval (overlapping with what previously have been called "print-sized characters"); Courier also had lower average standard deviation and maximum standard deviation compared to the other three fonts in the peak performance interval, implying that Courier really was performing worst in the peak performance interval.

The font size with highest noise tolerance was found to be font size 13, which had a noise tolerance of circa 5.14 %. It is concluded that the noise tolerance of Tesseract is circa 5.14+-1.07 % (average standard deviation of peak performance interval: circa 2.14, divided by two: 1.07), i.e. Tesseract can handle 6.21 % of the pixels in a image being different, and still be able to identify all characters in the image.

The analysis and this conclusion provide an answer to the research question (refer to section 1.2.2), given the scope in section 1.4:
Research question: "How much (quantification) image degradation can the OCR engine Tesseract handle on a Latin alphabet string before failing to recognize at least one character in the string, i.e. what are the recognition failure conditions of Tesseract when OCR input images are subjected to an implementation of image degradation modeling?"

The noise tolerance results from this thesis represent breaking points of the tested OCR engine, i.e. if the changed pixels percentage measurement (defined in

section 3.2.2) is below a certain threshold, one can be more confident about the resulting OCR result being correct. As mentioned in section 1.3, this knowledge can be used to improve OCR algorithms, thus being a future work possibility. The presented method may also be expanded to include more noise types and OCR engines.

Chattopadhyay et al[10], which was mentioned briefly in section 1.3.1, compare OCR engines when extracting text from video frames, i.e. from a television. Considering that salt and pepper noise may occur as a result of electronic circuit noise (refer to section 1.5), the noise tolerance result from this thesis might be applicable in the television OCR context. Knowing the noise tolerance of an OCR engine for real data in a certain context, in this case television text OCR, might help predict the OCR accuracy in the given context.

The presented experiment method could be expanded to use longer text strings or whole documents. The noise stopping criterion used in this thesis, an edit distance of 1, could be expanded to measure the OCR accuracy instead of noise, or perhaps a novel combination of the two.

Closing comment: finding out what fonts work best for a certain OCR engine gives an opportunity to select this font when developing e.g. bank checks or other documentation, that are to be recognized by the specified OCR engine.

References

[1] A. Kae, G. B. Huang, C. Doersch, and E. G. Learned-Miller, "Improving state-of-the-art ocr through high-precision document-specific modeling," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[2] H. Fujisawa, "A view on the past and future of character and document recognition," in *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, vol. 1, pp. 3–7, Sept 2007.

[3] S. V. Rice, G. L. Nagy, and T. A. Nartker, *Optical Character Recognition: An Illustrated Guide to the Frontier*. Norwell, MA, USA: Kluwer Academic Publishers, 1999.

[4] L. Vincent, "Google book search: Document understanding on a massive scale," in *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, vol. 2, pp. 819–823, Sept 2007.

[5] P. Ye and D. Doermann, "Learning features for predicting ocr accuracy," in *Pattern Recognition (ICPR), 2012 21st International Conference on*, pp. 3204–3207, 2012.

[6] H. S. Baird, "The state of the art of document image degradation modeling," in *In Proc. of 4 th IAPR International Workshop on Document Analysis Systems, Rio de Janeiro*, pp. 1–16, 2000.

[7] P. Ye and D. Doermann, "Document image quality assessment: A brief survey," in *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pp. 723–727, Aug 2013.

[8] R. Lins, "A taxonomy for noise in images of paper documents - the physical noises," in *Image Analysis and Recognition* (M. Kamel and A. Campilho, eds.), vol. 5627 of *Lecture Notes in Computer Science*, pp. 844–854, Springer Berlin Heidelberg, 2009.

[9] C. Liu, W. Freeman, R. Szeliski, and S. B. Kang, "Noise estimation from a single image," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1, pp. 901–908, June 2006.

[10] T. Chattopadhyay, P. Sinha, and P. Biswas, "Performance of document image ocr systems for recognizing video texts on embedded platform," in *Computational Intelligence and Communication Networks (CICN), 2011 International Conference on*, pp. 606–610, Oct 2011.

[11] R. Smith, "An overview of the tesseract ocr engine," in *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, vol. 2, pp. 629–633, 2007.

[12] C. Patel, A. Patel, and D. Patel, "Optical character recognition by open source ocr tool tesseract: A case study," *International Journal of Computer Applications*, vol. 55, no. 10, 2012.

[13] S. Jayaraman, S. Esakkirjan, and T. Veerakumar, *Digital Image Processing*. Tata McGraw-Hill Education, 2009.

[14] S. Dhanani and M. Parker, *Digital Video Processing for Engineers*. Newnes, 2012.

[15] M. de Castro Polastroa and N. F. Almeidac, "Ocr errors and their effects on computer forensics," in *The Sixth International Conference on Forensic Computer Science*, pp. 115–121, 2011.

[16] W. Bieniecki, S. Grabowski, and W. Rozenberg, "Image preprocessing for improving ocr accuracy," in *Perspective Technologies and Methods in MEMS Design, 2007. MEMSTECH 2007. International Conference on*, pp. 75–80, May 2007.

[17] M. S. Dhiman and P. Dr. A.J Singh, "Tesseract vs gocr a comparative study," *International Journal of Recent Technology and Engineering*, vol. 2, no. 4, pp. 80–83, 2013.

[18] K. Toh and N. Isa, "Noise adaptive fuzzy switching median filter for salt-and-pepper noise reduction," *Signal Processing Letters, IEEE*, vol. 17, pp. 281–284, March 2010.

[19] R. Chan, C.-W. Ho, and M. Nikolova, "Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization," *Image Processing, IEEE Transactions on*, vol. 14, pp. 1479–1485, Oct 2005.

[20] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *Image Processing, IEEE Transactions on*, vol. 13, pp. 600–612, April 2004.

[21] N. Damera-Venkata, T. Kite, W. Geisler, B. Evans, and A. Bovik, "Image quality assessment based on a degradation model," *Image Processing, IEEE Transactions on*, vol. 9, pp. 636–650, Apr 2000.

[22] H. Baird, "Document image defect models," in *Structured Document Image Analysis* (H. Baird, H. Bunke, and K. Yamamoto, eds.), pp. 546–556, Springer Berlin Heidelberg, 1992.

[23] H. Baird, "Document image defect models and their uses," in *Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on*, pp. 62–67, 1993.

[24] I. Guyon, R. M. Haralick, J. J. Hull, and I. T. Phillips, "Data sets for ocr and document image understanding research," in *In Proceedings of the SPIE - Document Recognition IV*, pp. 779–799, World Scientific, 1997.

[25] W. G. Zikmund, B. J. Babin, J. C. Carr, and M. Griffin, *Business Research Methods, 8th editon.* Cengage Learning, 2009.

Chapter 7

Appendix

7.1 Fonts tested, illustrated with the largest font size tested: 100

Figure 7.1: Test case: 100, Arial, "DEFY"

Figure 7.2: Test case: 100, Calibri, "DEFY"

Figure 7.3: Test case: 100, Courier, "DEFY"

Figure 7.4: Test case: 100, Georgia, "DEFY"

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
1	1																																			
2	0.2222	1																																		
3	0.5385	0.3077	1																																	
4	0.2222	0.1667	0.0769	1																																
5	0.4444	0.5000	0.4615	0.1250	1																															
6	0.6667	0.2222	0.3846	0	0.3333	1																														
7	0.4000	0.4000	0.4615	0.1000	0.4000	0.4000	1																													
8	0.1111	0.1667	0.1538	0	0.1250	0.1111	0.2000	1																												
9	0.5385	0.2308	0.6154	0.0769	0.3077	0.5385	0.4615	0.0769	1																											
10	0.2222	0.1667	0.1538	0.5000	0.2500	0.1111	0.2000	0	0.2308	1																										
11	0.4444	0.4444	0.3077	0.1111	0.3333	0.5556	0.4000	0.1111	0.4615	0.2222	1																									
12	0	0	0	0	0	0	0.2000	0.2500	0	0	0	1																								
13	0	0.1667	0.1538	0.2000	0.3750	0.1111	0.1000	0	0.0769	0.4000	0.1111	0	1																							
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1																						
15	0.2222	0.5714	0.2308	0	0.3750	0.4444	0.5000	0.1429	0.4615	0.2857	0.4444	0	0.1429	0	1																					
16	0.1111	0.1667	0.1538	0	0.3750	0	0.1000	0	0.1538	0.2500	0.1111	0	0.4000	0	0.1429	1																				
17	0.1111	0.1667	0.0769	0.2000	0.2500	0.2222	0.1000	0	0.0769	0	0.1111	0	0.2000	0	0.1429	0	1																			
18	0.1111	0	0	0.3333	0.1250	0	0	0	0.0769	0.2500	0.2222	0	0.2000	0	0	0.2500	0	1																		
19	0.3333	0	0.2308	0	0	0.3333	0.1000	0	0.2308	0	0.1111	0	0	0	0	0	0	0	1																	
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1																
21	0.2308	0	0.3077	0	0	0.2308	0.0769	0	0.3077	0	0.0769	0	0	0	0	0	0	0	0	0.2222	1															
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1														
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1													
24	0.3333	0	0.2308	0	0	0.3333	0.1000	0	0.2308	0	0.1111	0	0	0	0	0	0	0	0	0.6667	0.2222	0.3846	0	0.3333	1										
25	0.1000	0	0.0769	0	0	0.1000	0.1000	0	0.0769	0	0.1000	0	0	0	0	0	0	0	0	0.4000	0.4000	0.4615	0.1000	0.4000	0.4000	1									
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1111	0.1667	0.1538	0	0.1250	0.1111	0.2000	1									
27	0.2308	0	0.3077	0	0	0.2308	0.0769	0	0.3846	0	0.0769	0	0	0	0	0	0	0	0	0.5385	0.2308	0.6154	0.0769	0.3077	0.5385	0.4615	0.0769	1								
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2222	0.1667	0.1538	0.5000	0.2500	0.1111	0.2000	0	0.2308	1							
29	0.1111	0	0.0769	0	0	0.1111	0.1000	0	0.0769	0	0.1111	0	0	0	0	0	0	0	0	0.4444	0.4444	0.3077	0.1111	0.3333	0.5556	0.4000	0.1111	0.4615	0.2222	1						
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2000	0.2500	0	0	0	1						
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1667	0.1538	0.2000	0.3750	0.1111	0.1000	0	0.0769	0.4000	0.1111	0	1				
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1				
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1			
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1		
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 7.5: Occurrence matrix for the 36 strings used in the experiment, showing the correlation between them (refer to section 3.2.1 for index meanings). A correlation of 1 means they are the same string. Higher values means more correlation than lower values. Refer to figure 7.6 for calculation code.

```
function corrCoefT=stringCorr2(a,b)
a=double(a);
b=double(b);
mA=length(a);
nB=length(b);
if mA>nB
    B1=zeros(1,mA);
    B1(1:nB)=b;
    b=B1;
    numG=mA;
else,
    A1=zeros(1,nB);
    A1(1:mA)=a;
    a=A1;
    numG=nB;
end
A=zeros(128);
B=A;
for kk=1:max(mA,nB)
    if a(kk)>0,
        A(a(kk),a(kk))=A(a(kk),a(kk))+1;
    end
    if b(kk)>0,
        B(b(kk),b(kk))=B(b(kk),b(kk))+1;
    end
end
tp=find(A>1);
dBA=sum(A(tp))-length(tp);
tp=find(B>1);
dBB=sum(B(tp))-length(tp);
doubleEffect=min(dBA,dBB);
cc=sum(sum(and(A,B)))+doubleEffect;
corrCoefT=cc/numG;
```

Figure 7.6: MATLAB code used to calculate string correlation between string "a" and string "b", converting them to their ASCII representation before calculation.

Font sizes	**Arial**	**Calibri**	**Courier**	**Georgia**
1-10	0.369503	0.166141	0.208567	0.648861
11-20	2.38877	2.38772	1.68505	2.04739
21-30	1.61918	1.79833	1.51048	1.51166
31-40	1.1863	1.28478	1.26019	1.10477
41-50	0.980616	1.07371	1.08095	0.862193
51-60	0.810904	0.949014	0.943171	0.691033
61-70	0.704001	0.809025	0.869959	0.586887
71-80	0.612331	0.722761	0.80417	0.507948
81-90	0.562696	0.65605	0.740875	0.491451
91-100	0.539878	0.611439	0.687994	0.477818

Table 7.1: The data used in figure 4.1B.

1 0	35 3.42284	69 1.90625
2 0	36 3.26659	70 1.8044
3 0	37 3.2446	71 1.80363
4 0	38 3.18596	72 1.76312
5 0	39 3.16937	73 1.7581
6 0	40 3.05517	74 1.71721
7 0.15162	41 3.01235	75 1.69252
8 0.756944	42 2.91937	76 1.66088
9 1.18248	43 2.85108	77 1.6088
10 2.90702	44 2.78395	78 1.55247
11 4.12963	45 2.78549	79 1.5571
12 4.88812	46 2.69522	80 1.49769
13 5.13735	47 2.74923	81 1.4784
14 5.00617	48 2.6439	82 1.48418
15 4.86381	49 2.59414	83 1.43981
16 4.89198	50 2.60147	84 1.42323
17 4.92593	51 2.53742	85 1.38426
18 4.90278	52 2.51196	86 1.37269
19 4.86613	53 2.44483	87 1.32832
20 4.74537	54 2.42284	88 1.27855
21 4.4591	55 2.3777	89 1.30748
22 5.10687	56 2.31674	90 1.2392
23 4.91512	57 2.28511	91 1.23804
24 4.95062	58 2.24421	92 1.20602
25 4.6223	59 2.23495	93 1.20409
26 4.59491	60 2.18943	94 1.17284
27 4.47107	61 2.14082	95 1.15394
28 4.26157	62 2.11844	96 1.13503
29 4.0733	63 2.04128	97 1.1196
30 4.03048	64 2.02623	98 1.10957
31 3.98457	65 2.03511	99 1.11188
32 3.79745	66 1.95139	100 1.08642
33 3.55903	67 1.9267	
34 3.44792	68 1.91898	

Figure 7.7: The data used in figure 4.1C. Presented as three columns. The first value in each column is the font size (x-value) and second value for each column is the noise tolerance (y-value).